1 The Application of Beowulf-Class Computing to Computational Electromagnetics

DANIEL S. KATZ and TOM CWIK
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California, 91109

1.1 INTRODUCTION

Current computational developments at the Jet Propulsion Laboratory (JPL) are motivated by the NASA/JPL goal of reducing payload in future space missions while increasing mission capability through miniaturization of active and passive sensors, analytical instruments and communication systems. Typical system requirements include the detection of particular spectral lines or bands, associated data processing, and communication of the acquired data to other systems including the transmission of processed data to Earth via telecommunication systems. Each of these systems can include various electromagnetic components that require analysis and optimization over a complex design space. Specifications are being pushed by the demands of miniaturization as well as improved performance and/or greater fidelity in the imaging and telecommunication system. Because an experimental exploration of the design space is impractical, the use of reliable software design tools executing on high performance computers is being advanced. These tools use models based on the fundamental physics and mathematics of the component or system being studied, and they strive to have convenient turn-around times and interfaces that allow effective usage. These tools are then integrated into an optimization environment, and, using the available memory capacity and computational speed of high performance parallel platforms, simulation of the components to be optimized proceeds.

Electromagnetic components and systems at JPL can be selectively grouped into classes related to a) instruments and devices used for passive and active sensing typically in the infrared to optical portions of the spectrum; b) components for filtering and guiding of energy for telecommunication or sensor applications; and c) antennas used for telescopes in the millimeter wave portion of the spectrum and for telecommunication in the microwave spectrum. Design efforts in the instrument and device class include finite element modeling of light coupling structures in quantum well infrared photodetectors and frequency selective surface integral equation models of near-infrared filters. Component design includes a wide range of waveguide couplers, horns and dichroic plates for radio frequency telecommunication and for millimeter wave sensors. These components are typically modeled using mode matching methods. Antenna design for telescopes and telecommunication systems involves physical optics analysis of beam-waveguide systems. Additionally, recent advances have led to integrated environments where electromagnetic models are coupled with structural and thermal models to allow design simulations of millimeter-wave telescopes that include surface distortions due to thermal loads on the telescope resulting from deep space trajectories. Finally, integrated environments call for optimization methods to be inserted into the design process to assist the designer through multi-parameter design spaces.

To fully carry out the goal of integrated design in an environment that includes optimization, high computational speeds and memory capacities are essential. These requirements can most easily be met by using parallel computing platforms. A wide range of electromagnetic codes have been ported to or developed for parallel machines at JPL. Initial work involved porting finite difference time domain and integral equation codes to a succession of distributed memory machines. This effort resulted in an understanding of machine performance and data decomposition for electromagnetic models that scaled with the number of processors and memory available. Additional work involved the development of finite element methods, including unstructured mesh decomposition techniques on distributed memory machines. Currently, parallel adaptive mesh refinement algorithms are being developed to more efficiently solve these problems. The exploration of global optimization methods has also begun, using parallel genetic algorithms integrated with electromagnetic models for grating structures. The parallel platforms that have been used in these projects have consisted of either one-of-a-kind machines or relatively expensive (though powerful) commercial computers, with the cost or availability of the parallel machines limiting their usefulness. Recently, a new class of parallel machines has become available. These machines, known as Beowulf-class computers, are built from commodity-off-the-shelf components connected together with commodity-off-the-shelf networking hardware and can achieve sustained performance equal to 10-100 personal computers.

Hyglac is an early Beowulf-class system, currently located at JPL. It consists of 16 nodes interconnected by 100Base-T Fast Ethernet. Each node include a single Intel Pentium Pro 200 MHz microprocessor, 128 MBytes of

DRAM, 2.5 GBytes of IDE disk, and a PCI bus backplane. All nodes have a video card and a floppy drive, and a single monitor and keyboard are switched between the nodes. One node also has a CD-ROM drive. Because technology is evolving extremely quickly and price performance and price feature curves are also changing very quickly, no two Beowulfs ever look exactly alike. Of course, this is also because the components are almost always acquired from a mix of vendors and distributors. The power of de facto standards for interoperability of subsystems has generated an open market that provides a wealth of choices for customizing ones' own version of Beowulf, or just maximizing cost advantage as prices fluctuate among sources. A Beowulf-class system runs the Linux [1] operating system freely available over the net or in low-cost and convenient CD-ROM distributions. In addition, publicly available parallel processing libraries such as MPI [2] and PVM [3] are used to harness the power of parallelism for large application programs. A small Beowulf system such as Hyglac costs less than $20K (as of July 1998) including all incidental components such as low cost packaging, taking advantage of appropriate discounts.

This chapter discusses results obtained on two Beowulf systems. Hyglac consists of 16 nodes interconnected by a 16 port Bay Networks 28115/ADV 100Base-T Fast Ethernet switch. The network switch is built around a 1.6 Gbps switch fabric, thus allowing up to 8 simultaneous 100 Mbps streams between 8 pairs of nodes. Naegling, a Beowulf system at Caltech has a larger number (between 64 and 110, depending on the needs of various projects) of CPUs which are individually the same as those on Hyglac. It also has a small number of CPUs that contain 300 MHz Pentium II processors. It uses a network switch that consists of two 80-port Fast Ethernet switches, connected by a 4 Gbit/s link. Each switch has a backplane bandwidth that should be sufficient for the full 80 ports.

Previously, a suite of 6 application codes was studied on Hyglac [4]. These included the three electromagnetic applications that are discussed in this chapter, as well as other science and engineering applications. There was a wide range in the amount of data being communicated as well as the pattern of communication across processors, and it was determined that the amount of communication was the most important factor in predicting Beowulf performance for each application. Because the electromagnetic applications ran well on Hyglac, two of them were further studied on Naegling [5]. This chapter reprises some of that study, discusses an addition application, and attempts to provide more detailed analysis of the three applications.

All of the applications use MPI or PVM for communication between processors and run on other platforms. It is useful to examine some measured data from a few machine that are available at JPL to understand what part of the code performance is dependent on the code itself, and what part is dependant on the platform. Tab. 1.1 shows that the Beowulf systems (Hyglac and Naegling) have lower peak computation rates that the Cray systems (T3D and T3E-600), by factors of 1.5 to 3, similar memory sizes, and lower communication rates, by factors of 4 to 18. This implies that communications performance of the applications will be much worse that computational performance, as compared with the Cray systems, and explains why the amount of communication versus the amount of computation can be used to predict performance on the Beowulf systems.

Table 1.1 Comparison of measured data for four machines. Communications data is for application-to-application communication using MPI.

	Hyglac	Naegling	T3D	T3E-600
CPU Speed (MHz)	200	200	150	300
Peak Rate (MFLOP/s)	200	200	300	600
MEMORY (Mbyte)	128	128	64	128
Communication Latency (s)	150	322	35	18
Communication Throughput (Mbit/s)	66	78	225	1200

1.2 PHYSICAL OPTICS METHOD

The software described in this section [6] is used to design and analyze reflector antennas and telescope systems. It is based simply on a discrete approximation of the radiation integral [7]. This calculation replaces the actual reflector surface with a triangularly faceted representation so that the reflector resembles a geodesic dome. The Physical Optics (PO) current is assumed to be constant in magnitude and phase over each facet so the radiation integral is reduced to a simple summation. This program has proven to be surprisingly robust and useful for the analysis of arbitrary reflectors, particularly when the near-field is desired and the surface derivatives are not known.

The PO code has been developed and used at JPL over the last 30 years. Systems that were designed and analyzed using this code include the Deep Space Network (DSN) antennas (used for communication with spacecraft), and the Microwave Instrument for the Rosetta Orbiter (MIRO) antenna system (used to measure surface temperature gradient, outgassing, and temperature of gasses in the coma of the comet P/54 Wirtanen.)

The PO code (generally) considers a dual-reflector calculation (as can be seen in Fig. 1.1), which can be thought of as five sequential operations: (1) creating a mesh with N triangles on the first reflector; (2) computing the currents on the first reflector using the standard PO approximation; (3) creating a mesh with M triangles on the second reflector; (4) computing the currents on the second reflector by utilizing the currents on the first reflector as the field generator; and (5) computing the required field values by summing the fields from the currents on the second reflector. The most time-consuming part of the calculation is the computation of currents on the second reflector due to the currents on the first, since for N triangles on the first reflector, each of the M triangles on the second reflector require an N-element sum over the first.

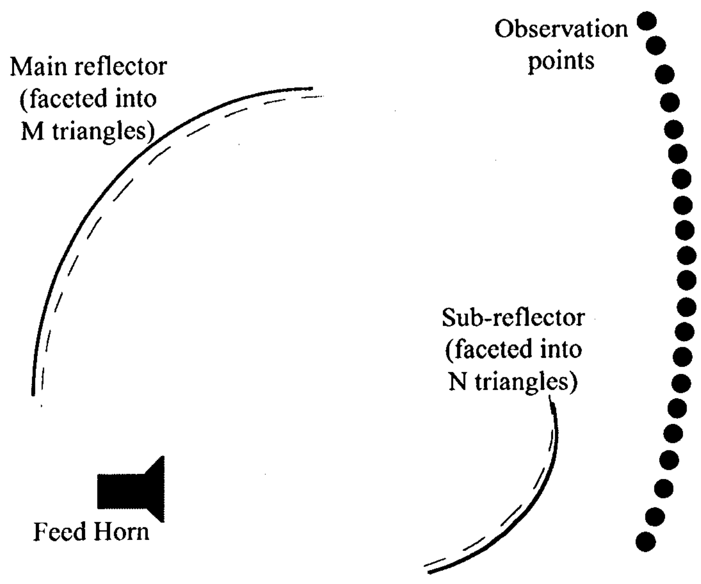
Main reflector (faceted into M triangles)
Observation points
Sub-reflector (faceted into N triangles)
Feed Horn

Fig. 1.1 The dual reflector Physical Optics problem, showing the source, the two reflectors, and the observation points.

Because of its simplicity, the algorithm has proven to be extremely easy to adapt to the parallel computing architecture of a modest number of large-grain computing elements such as are used in Beowulf-class machines. At this time, the code has been parallelized by distributing the M triangles on the second reflector over all the processors, and having all processors store the complete set of currents on the N triangles of the first reflector (though the computation of these currents is done in parallel.) Also, the calculation of observed field data has been parallelized. The main steps listed above are thus all performed in parallel, using simply a small number of global sums, and a single all-to-all gather. There are also sequential operations involved, such as I/O and the triangulation of the reflector surfaces, some of which potentially could be performed in parallel, but this would require a serious effort, and has not been done at this time.

While examining the performance of the PO code, two different compilers were compared (Gnu g77 and Absoft f77) on the Beowulf system. One set of indicative results from these runs are shown in Tab. 1.2 and 1.3. For this code, the Absoft compiler produced code that was approximately 30% faster, and this compiler was used hereafter.

It should be mentioned that the computation of the radiation integral in two places (in parts II and III, where the definition of the parts is given below) originally had code of the form:

```
CEJK = CDEXP(-AJ*AKR)
```

where `AJ = (0.D0,1.D0)`. This can be rewritten as:

```
CEJK = DCMPLX(DCOS(AKR),-DSIN(AKR))
```

On the T3D and T3E, these two changes led to improved results (the run-times were reduced by 35 to 40%.) When these changes were applied to the Beowulf code using the second compiler, no significant performance change was observed, leading to the conclusion that one of the optimizations performed by this compiler was similar to this hand-optimization.

Table 1.2 Timing results (in minutes) for PO code, for M=40,000, N=4,900, using the gnu g77 compiler.

Number of Processors	I	II	III	Total
1	0.0850	64.3	1.64	66.0
4	0.0515	16.2	0.431	16.7
16	0.0437	4.18	0.110	4.33

Table 1.3 Timing results (in minutes) for PO code, for M=40,000, N=4,900, using the Absoft f77 compiler.

Number of Processors	I	II	III	Total
1	0.0482	46.4	0.932	47.4
4	0.0303	11.6	0.237	11.9
16	0.0308	2.93	0.0652	3.03

The timings for each physical optics run are broken down into three parts. Part I is input I/O and triangulation of the main reflector surface, some of which is done in parallel. Part II is triangulation of the sub-reflector surface, evaluation of the currents on the sub-reflector, and evaluation of the currents on the main reflector. As stated previously, the triangulation of the sub-reflector and evaluation of the currents on those triangles is done redundantly, while evaluation of the currents on the main reflector is done in parallel. Part III is evaluation of far fields (parallel) and I/O (on only one processor).

It may be observed from Tab. 1.4, 1.5, and 1.6 that the Beowulf code performs slightly better than the T3D code, both in terms of absolute performance as well as scaling from 1 to 64 processors. (Tab. 1.4 and 1.5 contain results obtained on Hyglac, and Tab. 1.6 contains results obtained on Naegling.) This performance difference can be explained by the faster CPU on the Beowulf versus the T3D, and the very simple and limited communication not enabling the T3Ds faster network to influence the results. The scaling difference is more a function of I/O, which is both more direct and more simple on the Beowulf, and thus faster. By reducing this part of the sequential time, scaling performance is improved. Another way to look at this is to compare the results in the three tables. Clearly, scaling is better in the larger test case, in which I/O is a smaller percentage of overall time. It is also clear that the communications network used on Naegling is behaving as designed for the PO code running on 4, 16, or 64 processors. Since the majority of communication is single word global sums, this basically demonstrates that the network has reasonable latency for this code.

Table 1.4 Timing results (in seconds) for PO code, for M=40,000, N=400.

Number of Processors	Beowulf			T3D		
	I	II	III	I	II	III
1	3.19	230	56.0	14.5	249	56.4
4	1.85	57.7	14.2	8.94	62.5	14.7
16	1.52	14.6	3.86	8.97	16.6	4.13

Table 1.5 Timing results (in minutes) for PO code, for M=40,000, N=4,900.

Number of Processors	Beowulf			T3D		
	I	II	III	I	II	III
1	0.0482	46.4	0.932	0.254	48.7	0.941
4	0.0303	11.6	0.237	0.149	12.2	0.240
16	0.0308	2.93	0.0652	0.138	3.09	0.0749

Tab. 1.7, 1.8, and 1.9 show comparisons of complete run time for the 3 test problems sizes, for the Beowulf, T3D, and T3E-600 systems. These demonstrate good performance on the two Beowulf-class machines when compared with the T3D in terms of overall performance, as well as when compared with the T3E-600 in terms of price-performance. For all three test cases, the Beowulf scaling is better than the T3D scaling, but the results are fairly close for the largest test case, where the Beowulf being used is Naegling. This can be explained in large part by I/O requirements and timings on the various machines. The I/O is close to constant for all test cases over all machine sizes,

Table 1.6 Timing results (in minutes) for PO code, for M=160,000, N=10,000.

Number of Processors	Beowulf			T3D		
	I	II	III	I	II	III
4	0.0950	94.6	0.845	0.546	101	0.965
16	0.0992	23.9	0.794	0.463	25.6	0.355
64	0.0950	6.38	0.541	0.520	6.93	0.116

so in some way it acts as serial code that hurts scaling performance. The I/O is the fastest on Hyglac, and slowest on the T3D. This is due to the number of nodes being used on the Beowulf machines, since disks are NFS-mounted, and the more nodes there are, the slower the performance is using NFS. The T3D forces all I/O to travel through its Y-MP front end, which causes it to be very slow. Scaling on the T3D is generally as good as the small Beowulf, and faster than the large Beowulf, again due mostly to I/O. It may be observed that the speed-up of the second test case on the T3E is superlinear in going from 1 to 4 processors. This is probably caused by a change in the ratio of some of the size of some of the local arrays to the cache size dropping below 1.

Table 1.7 Timing results (in seconds) for complete PO code, for M=40,000, N=400.

Number of Processors	Beowulf(Hyglac)	Cray T3D	Cray T3E-600
1	289	320	107
4	73.8	86.1	29.6
16	20.0	29.2	8.36

Table 1.8 Timing results (in minutes) for complete PO code, for M=40,000, N=4,900.

Number of Processors	Beowulf(Hyglac)	Cray T3D	Cray T3E-600
1	47.4	49.4	18.4
4	11.9	12.6	4.43
16	3.03	3.30	1.14

A hardware monitoring tool was used on the T3E to measure the number of floating point operations in the M=40,000, N=4,900 test case as 1.32×10^{11} floating point operations. This gives a rate of 46, 44, and 120 MFLOP/s on one processor of the Beowulf, T3D, and T3E-600 respectively. These are fairly

Table 1.9 Timing results (in minutes) for complete PO code, for M=160,000, N=10,000.

Number of Processors	Beowulf(Naegling)	Cray T3D	Cray T3E-600
4	95.5	102	35.1
16	24.8	26.4	8.84
64	7.02	7.57	2.30

good (23, 29, and 20% of peak, respectively) for RISC processors running FORTRAN code.

A few tests have been run for small test cases using 300 MHz Pentium II nodes on Naegling. They have shown a speedup of 40% for the computation part of the code over the 200 MHz Pentium Pro nodes.

1.3 FINITE-DIFFERENCE TIME-DOMAIN METHOD

The software described in this section can be used for finding antenna patterns, electromagnetic scattering from targets, fields within small electronic circuits and boards, as well as in bioelectromagnetic and photonic simulations. The method directly produces results in the time domain (transient fields), and by some fairly simple post-processing (Fourier transforms), can produce results in the frequency domain at a number of frequencies. Models can include a variety of materials including inhomogeneous, anisotropic and non-linear materials.

Finite-differencing was introduced by Yee in the mid 1960s as an efficient way of solving Maxwell's time-dependent curl equations [8]. His method involved sampling a continuous electromagnetic field in a finite region at equidistant points in a spatial lattice, and at equidistant time intervals. Spatial and time intervals have been chosen to avoid aliasing and to provide stability for the time-marching system [9]. The propagation of waves from a source, assumed to be turned on at time $t = 0$, is computed at each of the spatial lattice points by using the finite difference equations to march forward in time. This process continues until a desired final state has been reached (often the steady state). This method has been demonstrated to be accurate for solving for millions of field unknowns in a relatively efficient manner on sequential and parallel computers.

This version of the FDTD algorithm uses a uniform Cartesian grid, and describes the object being studied as a combination of cubic cells and square faces. More complex versions exist for more accurately modeling curved surfaces and thin features, such as wires or composite materials, but these versions are often based on a Cartesian code, with small modification in regions where objects do not align themselves with the coarse grid. The vast majority of both the operations required and the physical region that is modeled is a Cartesian grid, and thus, this is the key work to be run in parallel efficiently.

This code is second order in both space and time. Using the sampling method chosen by Yee, updating a given field component requires knowledge of that same component one time step previously, as well as knowledge of additional field components within a planar region with size one square spatial cell. A typical update has a cross-shaped stencil, and looks like:

$$\begin{aligned} A_z(x,y,z,t) &= C_a(x,y,z) \times A_z(x,y,z,t-\Delta t) + \\ & C_b(x,y,z) \times \begin{pmatrix} B_x(x,y+\Delta x,z,t-\Delta t/2)- \\ B_x(x,y-\Delta x,z,t-\Delta t/2)+ \\ B_y(x+\Delta x,y,z,t-\Delta t/2)- \\ B_y(x-\Delta x,y,z,t-\Delta t/2) \end{pmatrix} \end{aligned} \quad (1.1)$$

where A_z and (B_x, B_y) are field components (either electric or magnetic), and C_a and C_b are simply coefficients that are not functions of time.

The parallelization is done in a very straightforward manner; by decomposing the 3-dimensional physical domain being studied over the processors using a 2-dimensional decomposition (over x and y). All values of z for a given (x,y) are located on the same processor. As the update stencil requires data one cell away from the component being updated in each direction, at the edges of the decomposed regions, an additional plane of cells from a neighboring region is needed. As neighboring regions are located on neighboring processors, this introduces communications. The data that is required to be communicated is referred to a ghost cell data (or ghost cells). This is illustrated in Fig. 1.2 for a distribution on a 4 by 4 grid of processors.

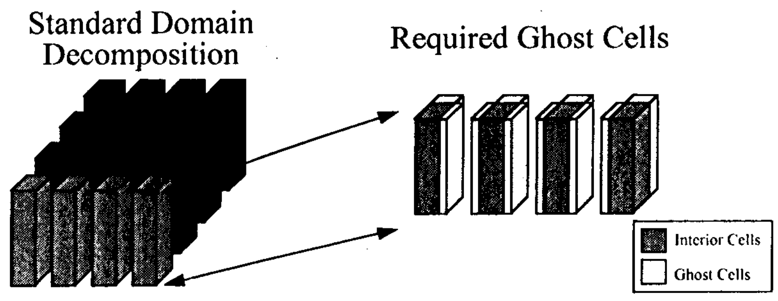

Fig. 1.2 The relation between the 2-D decomposition of the 3-D grid and the required ghost cell communication.

All FDTD results that are shown in this section use a fixed size local (per processor) grid, of 69 × 69 × 76 cells. The overall grid sizes therefore range from 69 × 69 × 76 to 552 × 552 × 76 (on 1 to 64 processors). (All Beowulf results are from Naegling.) This is the largest local problem size that may be solved on the T3D, and while the other machines have more local memory and could solve larger problems, it seems more fair to use the same amount of local work for these comparisons. In general, the FDTD method requires 10 to 20 points per wavelength for accurate solutions, and a boundary region of 10 to 20 cells in each direction is also needed. These grid sizes therefore correspond to scattering targets ranging in size from 5 × 5 × 5 to 53 × 53 × 5 wavelengths.

Both available compilers were used on the Beowulf version of the FDTD code. While the results are not tabulated in this paper, the gnu g77 compiler produced code which ran faster than the code produced by the Absoft f77 compiler. However, the results were just a few percent different, rather than on the scale of the differences shown by the PO code. All results shown here are from the gnu g77 compiler.

Tab. 1.10 shows results on various machines and various numbers of processors in units of CPU seconds per simulated time step. Complete simulations might require hundreds to hundreds of thousands time steps, and the results can be scaled accordingly, if complete simulation times are desired. Results are shown broken into computation and communication times, where communication includes send, receive, and buffer copy times.

Table 1.10 Timing results (in computation - communication CPU seconds per time step) for FDTD code, for fixed problem size per processor of 69 × 69 × 76 cells.

Number of Processors	Beowulf	Cray T3D	Cray T3E-600
1	2.44 - 0.0	2.71 - 0.0	0.851 - 0.0
4	2.46 - 0.097	2.79 - 0.026	0.859 - 0.019
16	2.46 - 0.21	2.79 - 0.024	0.859 - 0.051
64	2.46 - 0.32	2.74 - 0.076	0.859 - 0.052

It is clear that the Beowulf and T3D computation times are comparable, while the T3E times are about 3 times faster. This is reasonable, given the relative clock rates (200, 150, and 300 MHz) and peak performances (200, 150, 600 MFLOP/s) of the CPUs. As with the PO code, the T3D attains the highest fraction of peak performance, the higher clock rate of the Beowulf gives it a slightly better performance than the T3D, and the T3E obtains about the same fraction of peak performance as the Beowulf. As this code has much more communication that the PO code, there is a clear difference of an order of magnitude between the communication times on the Beowulf and the T3D and T3E. However, since this is still a relatively small amount of communication as compared with the amount of computation, it doesn't really effect the overall results.

It should also be noted that the use of 300 MHz Pentium II nodes produces a speed-up of 20% for the computation part of the code. This improves the Beowulf as compared with the Cray platforms. If the delivered performance of PC-class CPUs continues to increase faster than that of workstation CPUs, future Beowulf- class machines will become more and more competitive with vendor-produced parallel computers.

1.4 FINITE-ELEMENT INTEGRAL-EQUATION COUPLED METHOD

The finite element modeling software (discussed in greater detail in [10], [11]) begins with mesh data constructed on a workstation using a commercially available CAD meshing package. Because the electromagnetic scattering simulation is an open region problem (scattered fields exist in all space to infinity), the mesh must be truncated at a surface that maintains accuracy in the modeled fields, and limits the volume of free space that is meshed. Local, absorbing boundary conditions can be used to truncate the mesh, but these may be problematic because they become more accurate as the truncating surface is removed from the scatterer, requiring greater computational expense, and they may be problem dependent. The approach used in this section solves the three-dimensional vector Helmholtz wave equation, using a coupled finite element - integral equation method. A specific integral equation (boundary element) formulation that efficiently and accurately truncates the computational domain is used. A partitioned system of equations results from the combination of discretizing the volume in and around the scatterer using the finite element method, and discretizing the surface using the integral equation method. This system of equations is solved using a two-step solution, combining a sparse iterative solver and a dense factorization method. The matrix equation assembly, solution, and the calculation of observable quantities are all computed in parallel, utilizing varying number of processors for each stage of the calculation.

In this section, one of the three codes that make up the complete finite-element integral equation electromagnetic analysis package is discussed. This software package was originally implemented in parallel on the Cray T3D massively parallel processor using both Cray Adaptive FORTRAN (CRAFT) compiler constructs to simplify portions of the code that operate on the irregular data to build the matrix problem, and optimized message passing constructs on portions of the code that operate on regular data and require optimum machine performance to solve the matrix problem. The complete discussion of the parallel algorithm is given in [12].

This software is used similarly to the finite-difference software, except it computes results in the frequency domain, rather than in the time domain. It is also much more simple to input complex geometric structures into this code than into the FDTD code, by using commonly available commercial CAD packages.

The complete finite-element code (PHOEBUS) builds a large sparse matrix and solves it for many right hand sides. The matrix equation is:

$$\left| \begin{array}{ccc} \mathbf{K} & \mathbf{C} & \mathbf{0} \\ \mathbf{C}^{\dagger} & \mathbf{0} & \mathbf{Z_0} \\ \mathbf{0} & \mathbf{Z_M} & \mathbf{Z_J} \end{array} \right| \left| \begin{array}{c} \mathbf{H} \\ \mathbf{M} \\ \mathbf{J} \end{array} \right| = \left| \begin{array}{c} \mathbf{0} \\ \mathbf{0} \\ \mathbf{V_i} \end{array} \right| \tag{1.2}$$

where the symbol † indicates the adjoint of a matrix. Both **K** and **C** are sparse, $\mathbf{Z_0}$ is tri-diagonal, and $\mathbf{Z_M}$ and $\mathbf{Z_J}$ are banded. In particular the system is complex, non-symmetric, and non-Hermitian. The sparsity of the system is shown in Fig. 1.3 for a case with only several hundred finite element unknowns. For larger, representative cases, the number of finite element unknowns will grow into hundreds of thousands while the number of columns in **C** will be several hundred to several thousand.

The solution to this matrix equation system is completed in two steps. Initially **H** in the first equation in (1.2) is written as $\mathbf{H} = -\mathbf{K}^{-1}\mathbf{C}\mathbf{M}$ and substituted into the second equation resulting in

$$\left| \begin{array}{cc} \mathbf{Z_K} & \mathbf{Z_0} \\ \mathbf{Z_M} & \mathbf{Z_J} \end{array} \right| \left| \begin{array}{c} \mathbf{M} \\ \mathbf{J} \end{array} \right| = \left| \begin{array}{c} \mathbf{0} \\ \mathbf{V_i} \end{array} \right| \tag{1.3}$$

where $\mathbf{Z_K} = -\mathbf{C}^{\dagger}\mathbf{K}^{-1}\mathbf{C}$. This relatively small system is then solved directly for **M** and **J** . By solving the system in two steps, the interior solution is decoupled from the incident field $\mathbf{V_i}$, allowing for efficient solutions when many excitation fields are present as in monostatic radar cross section simulations. This section only discusses the code used in the first step, as the code used in the second step is quite straightforward.

The relative numbers of unknowns in **H** and **M** (or **J**) makes the calculation of $\mathbf{K}^{-1}\mathbf{C}$ the major computational expense. This operation is the solution of a system of equations, $\mathbf{KX} = \mathbf{C}$, where **C** is a rectangular matrix with a potentially large number of columns in the case of electrically large scatterers. The solution is accomplished by using a symmetric variant of the quasi-minimum residual (QMR) iterative algorithm. The resulting overall matrix (1.3) is treated as being dense, and the solution of this second system is accomplished via a direct dense LU decomposition, since its size is relatively small.

After the steps of building and distributing the matrix have been completed on a workstation or one processor of a parallel computer, or multiple processors of a parallel computer using a data parallel language, the solution of the matrix for each of the right hand sides is done on the parallel computer using a code written with a message passing model. This portion of the overall

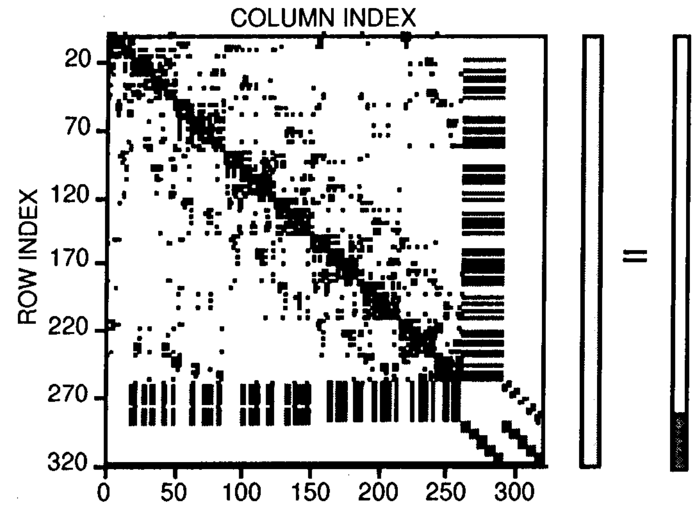

Fig. 1.3 Scatter plot graphically showing structure of system of equations. Darkened spaces indicate non-zero matrix entries.

problem usually requires over 98% of the complete problem time, and again, it is the work done by the code considered in this section.

After the matrix is been built, reordered to minimize and equalize row bandwidth, and distributed into files, these files are read into the message passing matrix solution code. A block-like iterative scheme (quasi-minimal residual) is used for the matrix solve, in which a matrix-vector multiply is the dominant component. Fig. 1.4 illustrates the process performed on a single processor in the matrix-vector multiply.

Because the matrix has been distributed in a one-dimensional processor grid, it makes sense to distribute the vectors similarly. The processor doing each portion of the multiply must acquire portions of the vector from its neighboring processors, as determined by the column extent of non-zeros in its portion of the matrix, and then perform the floating point operations of the multiply. The resultant vector is local to this processor, and no further communication is required.

Tab. 1.11 (and 1.12) show performance data for this QMR code, running on 16 (and 64) processors, and solving a matrix problem formed from a model of a dielectric cylinder, with radius = 1 cm, height = 10 cm, permittivity = 4.0, and frequency = 5.0 GHz. The matrix is complex, with 43,791 (and 101,694) rows having an average of 16 non-zero elements per row. The physical problem

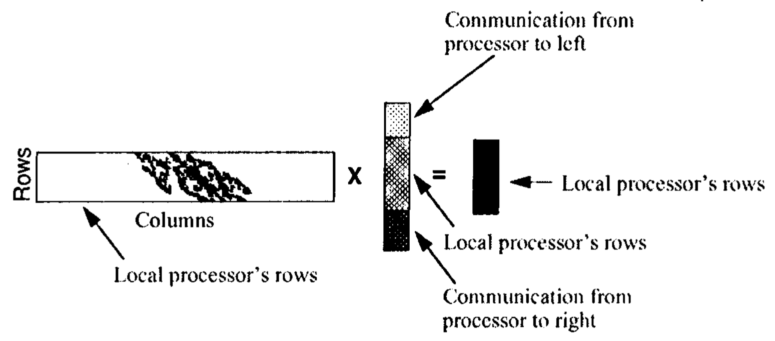

Fig. 1.4 A representation of the work required in each processor to perform a matrix-vector multiply.

requires solving this matrix for 116 right hand sides. The items in this table are measurements of the time spent in each part of the matrix solution for all right hand sides, in CPU seconds. Again, the two Beowulf compilers produced code that ran with nearly identical speed.

Table 1.11 CPU seconds required for each portion of the matrix solution for 116 right hand sides, 43,791 edges, using 16 processors.

	T3D(shmem)	T3D(MPI)	Beowulf(MPI)
Matrix-Vector Multiply Computation	1290	1290	1502
Matrix-Vector Multiply Communication	114	272	1720
Other Work	407	415	1211
Total	1800	1980	4433

The computation in the matrix-vector multiply is 55% faster on the Beowulf CPU than on the T3D CPU. This is due to a combination of increased clock speed and increased cache size. The communication is about 10 times slower on the Beowulf compared with the T3D (both using MPI), which is reasonable, given that this problem requires a large number of messages of both small and large length, as well as a fairly large number of small-size global sums. With messages of this size, this code is obtaining as much throughput as is possible with MPI. All of the global sums are included in the data of the Other Work row, which also contains a large number of vector-vector operations (including dot products, norms, scales, copies.) The increase in time of this work from

Table 1.12 CPU seconds required for each portion of the matrix solution for 116 right hand sides, 101,694 edges, using 64 processors.

	T3D(shmem)	T3D(MPI)	Beowulf(MPI)
Matrix-Vector Multiply Computation	868	919	1034
Matrix-Vector Multiply Communication	157	254	2059
Other Work	323	323	923
Total	1348	1496	4016

the T3D to the Beowulf can be viewed as a function of decreased memory-bandwidth, as there is almost no cache reuse in this work.

Overall, this problem is almost 3 times slower on the Beowulf than the T3D, due to a combination of communication speed, memory-bandwidth, and amount of communication. As with the previous codes, limited studies have been performed using 300 MHz Pentium II nodes. They have shown a speedup of 40% for the computation part of the code. Even with this factor, this code has decent but not outstanding price-performance, compared with the current value of a T3D. Faster communication is needed for the Beowulf version of the code to be truly comparable.

1.5 CONCLUSIONS

The intent of this chapter was to discuss the Beowulf class of computers, focusing on communication performance, since the computation performance of the personal computer CPU which forms the building block of the Beowulf is well understood, and to reach some conclusions about what this performance implies regarding the feasibility of using this type machine to run electromagnetic codes in an institutional science and engineering environment, such as at JPL.

This chapter has shown that for both parallel calculation of the radiation integral and parallel finite-difference time-domain calculations, a Beowulf-class computer provides slightly better performance that a Cray T3D, at a much lower cost. The limited amount of communication in the physical optics code defines it as being in the heart of the regime in which Beowulf-class computing is appropriate, and thus it makes a good test code for an examination of code performance and scaling, as well as an examination of compiler options and other optimizations. The FDTD code contains more communication, but the amount is still fairly small when compared with the amount of computation, and this code is a good example of domain decomposition PDE solvers. (The timing results from this code show trends that are very similar to the results of other domain decomposition PDE solvers that have been examined at JPL.)

The physical optics software had the best performance, primarily due to its almost embarrassingly parallel nature. The limited communications required by the software led to very good overall performance, due mostly to the CPU speed of the Beowulf being 33% faster than that of the T3D. This code is superior in both absolute performance and price-performance on the Beowulf than on the T3D.

The electromagnetic finite-element software performed similarly to the finite-difference software, in that computation was faster, and communication was slower. The unique aspect of this code is the large amount of BLAS 1-type operations that are performed with data moved from main memory, rather than from cache. This work is substantially slower than similar work on the T3D. The reason for this is a combination of poorer memory-CPU throughput, and lack of optimized BLAS routines for the JPL Beowulf. BLAS routines which have been optimized for the Pentium Pro CPU under Linux have recently been released, and they will be examined in the near future. These new routines are expected to improve the performance on the PHOEBUS code. Overall, even without this change, this code has acceptable performance on the Beowulf.

An interesting observation is that for Beowulf-class computing, using commodity hardware, the user also must be concerned with commodity software, including compilers. As compared with the T3D, where Cray supplies and updates the best compiler it has available, the Beowulf system has many compilers available from various vendors, and it is not clear that any one always produces better code than the others. In addition to the compilers used in this paper, at least one other exists (to which the authors did not have good access.) The various compilers also accept various extensions to FORTRAN, which may make compilation of any given code difficult or impossible without re-writing on some of it, unless of course the code was written strictly in standard FORTRAN 77 (or FORTRAN 90), which seems to be extremely uncommon.

It is also interesting to notice that the use of hand-optimizations produces indeterminate results in the final run times, again depending on which compiler and which machine is used. Specific compiler optimization flags have not been discussed in this paper, but the set of flags that was used in each case were those that produced the fastest running code, and in most but not all cases, various compiler flag options produced greater variation in run times that any hand optimizations. The implication of this is that the user should try to be certain there are no gross inefficiencies in the code to be compiled, and that it is more important to choose the correct compiler and compiler flags.

Overall, this chapter has examined and validated the choice of a Beowulf-class computer for the physical optics application (and other similar low-communication applications), the finite-difference time-domain application (and other domain decomposition PDE solvers), and the finite-element application (and other similar sparse iterative algorithms). It has examined performance of these codes in terms of comparison with the Cray T3D and T3E, scaling, and compiler issues, and pointed out some features of which users of Beowulf-systems should be aware.

Acknowledgments

The work described was performed at the Jet Propulsion Laboratory, California Institute of Technology under contract with the National Aeronautics and Space Administration. The Cray T3D supercomputer used in this investigation was provided by funding from the NASA Offices of Earth Science, Aeronautics, and Space Science. Part of the research reported here was performed using the Beowulf system operated by the Center for Advanced Computing Research at Caltech; access to this facility was provided by Caltech. Access to the Cray T3E-600 was provided by the Earth and Space Science (ESS) component of the NASA High Performance Computing and Communication (HPCC) program. The authors also wish to acknowledge Charles Norton's help with LaTeX.

REFERENCES

1. K. Husain, T. Parker, et al., *Red Hat Linux Unleashed*, Sams Publishing, Indianapolis, Ind., 1996.

2. M. Snir, S. W. Otto, S. Huss-Lederman, D. W. Walker, and J. Dongarra, *MPI: The Complete Reference*, The MIT Press, Cambridge, Mass., 1996.

3. A. Giest, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam, *PVM: A Users' Guide and Tutorial for Networked and Parallel Computing*, The MIT Press, Cambridge, Mass., 1994.

4. D. S. Katz, T. Cwik, B. H. Kwan, J. Z. Lou, P. L. Springer, T. Sterling, and P. Wang, An Assessment of a Beowulf System for a Wide Class of Analysis and Design Software, *Advances in Engineering Software,* **26(3-6),** pp. 451- 461 (July 1998).

5. D. S. Katz, T. Cwik, and T. Sterling, An Examination of the Performance of Two Electromagnetic Simulations on a Beowulf-Class Computer, *High Performance Computing Systems and Applications*, J. Schaeffer and R. Unrau, Editors, Kluwer Academic Press, in press.

6. W. A. Imbriale and T. Cwik, A Simple Physical Optics Algorithm Perfect for Parallel Computing Architecture, *10th Annual Review of Progress in Appl. Comp. Electromag.*, 434-441, Monterey, Cal., 1994.

7. W. A. Imbriale and R. Hodges, Linear Phase Approximation in the Triangular Facet Near-Field Physical Optics Computer Program, *Applied Computational Electromagnetics Society Journal,* **6** (1991).

8. K. S. Yee, Numerical Solution of Initial Boundary Value Problems involving Maxwell's Equations in Isotropic Media, *IEEE Transactions on Antennas and Propagation,* **14(5)**, 302-307 (1966).

9. A. Taflove, *Computational Electrodynamics: The Finite-Difference Time-Domain Method,* Artech House, Norwood, Mass., 1995.

10. T. Cwik, C. Zuffada, and V. Jamnejad, Efficient Coupling of Finite Element and Integral Equation Representations for Three-Dimensional Modeling, *Finite Element Software for Microwave Engineering,* T. Itoh, G. Pelosi, P. Silvester, Editors, John Wiley and Sons, Inc., New York, 1996.

11. T. Cwik, C. Zuffada, and V. Jamnejad, Modeling Three-Dimensional Scatterers Using a Coupled Finite Element - Integral Equation Representation, *IEEE Transactions on Antennas and Propagation,* **44(4)**, 453-459, (1996).

12. T. Cwik, D. S. Katz, C. Zuffada and V. Jamnejad, The Application of Scalable Distributed Memory Computers to the Finite Element Modeling of Electromagnetic Scattering and Radiation, *International Journal on Numerical Methods in Engineering,* **41(4)**, 759-776, (1998).